Module 5 – Programming TSQL

Overview

The main purpose of this lab is to familiarise yourself with programming within TSQL.

- Appropriately use the GO batch separator
- Assign variables and identify their scope
- Use IF and ELSE
- Use BEGIN and END

Objectives

At the end of this lab, you will be able to:

- Use the following commands with TSQL
  - GO
  - IF and ELSE
  - BEGIN and END
- Identify the need for variables and scope of the variables

Setup: Launch SQL Server Management Studio

1. On the Start menu, click All Programs, click Microsoft SQL Server 2014 and then click SQL Server Management Studio.
2. The Microsoft SQL Server Management Studio window opens, and then the Connect to Server dialog box will appear.
3. In the Connect to Server dialog box, click Connect to accept the default settings.
4. On the toolbar, click New Query, and either select the QATSQLPLUS database in the Available Databases box, or type USE QATSQLPLUS in the query window.
5. If you wish, you may save your queries to My Documents or the desktop. All modules are separate and you will not require any queries from this module in any later module.

Exercise 1: Issue Fixing

In this exercise, you will be given a portion of code that needs to be fixed.

The main tasks for this exercise are as follows:

1. Examine the given code and fix so that:
   - The first loop works correctly
   - The view is deleted if it already exists
   - The view is created

Task 1 : Review code

1. Review the code below:

```
DECLARE @StartDT datetime = GETDATE()
DECLARE @Vendor VARCHAR(50)
PRINT @StartDT
GO

DECLARE @X INT = 0
WHILE @X < 100
    BEGIN
        PRINT @X
        SET @X = @X + 1
    END
SET @EndDT datetime = GETDATE()
SELECT @StartDT, @EndDT

IF EXISTS (SELECT * FROM sysobjects WHERE Name = 'NewView')
    DROP VIEW dbo.NewView

CREATE VIEW dbo.NewView AS
    SELECT * FROM dbo.Delegate
GO
```

2. Diagnose why the errors are being shown.
3. Open the file “d:\qatsqlplus\M05E01 Batch Review.sql” and correct the errors.
4. Run the query to confirm no errors are returned and the query executes successfully.
5. The query window can be closed, and optionally saved.

Exercise 2: Assign value

In this exercise, you will be return the number of delegates into a variable.

The main tasks for this exercise are as follows:

1. Declare a variable.
2. Assign a value from a query to the variable.
3. Print the variable result.

Task 1: Write the code

1. Open a new query window in SSMS.
2. Declare the variable @TotalDelegates of type INT.
3. Assign the @TotalDelegates the number of delegates from the dbo.Delegates table.
4. Print the @TotalDelegates variable value. The result should be 32.

Exercise 3: Value Checking

In this exercise, you'll find produce batch of code that will use a variable to check whether a query should run or a message be returned.

The main tasks for this exercise are as follows:

1. Review an existing piece of code and trial with a couple of possible variable values.
2. Add code to prevent the query executing if a NULL value is given to the variable, instead returning a message.
3. Add code to prevent the query executing if not existent Vendor value is given to the variable, instead returning a message.

Task 1: Trialing the code

1. Open the file "d:\qatsqlplus\M05E03 If Else.sql".
2. Test the query with the @Vendor set to 'QA'. The query should return 4 rows.
3. Test the query with the @Vendor set to NULL. The query should return no rows.
4. Keep the query window open for the following tasks.

Task 2: Checking for NULL value

1. Update the query to check for a NULL value after the SET statement. An IF statement should be used and a message returned if the value is NULL. The original query should be performed if the value is not NULL.
2. Test the query with the @Vendor set to 'QA'. The query should return 4 rows.
3. Test the query with the @Vendor set to NULL. A message should be shown.
4. Keep the query window open for the following tasks.

Task 3: Checking for non-existent Vendors

1. Update the query to also check for a non-existent Vendor names.
    - Use the code below for the IF statement:

      ```
      IF NOT EXISTS (SELECT * FROM dbo.Vendor
            WHERE VendorName = @Vendor)
      ```

    - This should form an IF within the original ELSE section.

      ```
      IF @Vendor IS NULL
            ….
      ELSE
            IF NOT EXISTS (SELECT * FROM dbo.Vendor
                  WHERE VendorName = @Vendor)
                  ....
            ELSE
                  ….
      ```

2. Test the query with the @Vendor set to 'QA'. The query should return 4 rows.
3. Test the query with the @Vendor set to NULL. A message should be shown.
4. Test the query with the @Vendor set to 'AQ'. A message should be shown.
5. Keep the query window open for the following tasks.

Task 4 : Checking for non-existent Vendors (again)

1. Restructure the previous query to use the RETURN statement.
    - The return statement should be placed after each message is shown
    - The return statement stops the execution of following code
    - The IF statements should follow each other instead of being embedded
2. Test the query with the @Vendor set to 'QA'. The query should return 4 rows.
3. Test the query with the @Vendor set to NULL. A message should be shown.
4. Test the query with the @Vendor set to 'AQ'. A message should be shown.
5. Close the query and save if you would like.

Answers

The answers below are for example only. Coding style and order of columns should not matter.

Ex 1 Task 1

```
DECLARE @StartDT datetime = GETDATE()
DECLARE @EndDT datetime --@EndDT was not declared
DECLARE @Vendor VARCHAR(50)
PRINT @StartDT
--GO (this GO is not required and stops the variables being
--used in the loop that follows)

DECLARE @X INT = 0
WHILE @X < 100
  BEGIN
    PRINT @X
    SET @X = @X + 1
  END
SET @EndDT = GETDATE()
SELECT @StartDT, @EndDT
GO --GO optional but good practice

IF EXISTS (SELECT * FROM sysobjects WHERE Name = 'NewView')
  DROP VIEW dbo.NewView
GO --GO required before the CREATE VIEW statement

CREATE VIEW dbo.NewView AS
  SELECT * FROM dbo.Delegate
GO --GO optional but good practice
```

Ex 2 Task 1

```
--TASK 1:
DECLARE @TotalDelegates INT
SELECT @TotalDelegates = COUNT(*)
  FROM dbo.Delegate
PRINT @TotalDelegates
```

Ex 3 Task 1

```
--TASK 1(a):
DECLARE @Vendor VARCHAR(100)
SET @Vendor = 'QA'
SELECT *
  FROM dbo.Course AS C
    INNER JOIN dbo.Vendor AS V
      ON C.VendorID = V.VendorID
  WHERE
    VendorName = @Vendor
GO

--TASK 1(b):
DECLARE @Vendor VARCHAR(100)
SET @Vendor = NULL
SELECT *
  FROM dbo.Course AS C
    INNER JOIN dbo.Vendor AS V
      ON C.VendorID = V.VendorID
  WHERE
    VendorName = @Vendor
GO
```

Ex 3 Task 2

```
--TASK 2:
DECLARE @Vendor VARCHAR(100)
SET @Vendor = NULL

IF @Vendor IS NULL
   PRINT 'Vendor cannot be NULL'
ELSE
   SELECT *
     FROM dbo.Course AS C
      INNER JOIN dbo.Vendor AS V
        ON C.VendorID = V.VendorID
     WHERE
      VendorName = @Vendor
GO
```

Ex 3 Task 3

```
--TASK 3:
DECLARE @Vendor VARCHAR(100)
SET @Vendor = 'AQ'

IF @Vendor IS NULL
  PRINT 'Vendor cannot be NULL'
ELSE
  IF NOT EXISTS (SELECT * FROM dbo.Vendor WHERE VendorName=@Vendor)
    PRINT 'Vendor name not found'
  ELSE
    SELECT *
      FROM dbo.Course AS C
        INNER JOIN dbo.Vendor AS V
         ON C.VendorID = V.VendorID
      WHERE
        VendorName = @Vendor
GO
```

Ex 3 Task 4

```
--TASK 4:
DECLARE @Vendor VARCHAR(100)
SET @Vendor = 'QA'

IF @Vendor IS NULL
  BEGIN
    PRINT 'Vendor cannot be NULL'
    RETURN
  END

IF NOT EXISTS (SELECT * FROM dbo.Vendor WHERE VendorName=@Vendor)
  BEGIN
    PRINT 'Vendor name not found'
    RETURN
  END

SELECT *
  FROM dbo.Course AS C
    INNER JOIN dbo.Vendor AS V
      ON C.VendorID = V.VendorID
  WHERE
    VendorName = @Vendor
GO
```